Hello! This looks like a solid set of Oracle PL/SQL exercises. It covers everything from basic triggers to package state management and bulk processing.

I've written the solutions for each of the 6 tasks below. Each block is self-contained and follows the specific logic and naming conventions you provided.

### 1. Trigger: Change Audit

This trigger monitors the clients table. It uses the INSERTING and UPDATING predicates to determine the operation and logs the changes into the client\_audit\_log.

SQL

CREATE OR REPLACE TRIGGER trg\_clients\_audit  
AFTER INSERT OR UPDATE ON clients  
FOR EACH ROW  
BEGIN  
 INSERT INTO client\_audit\_log (audit\_id, audit\_ts, operation, message)  
 VALUES (  
 seq\_audit.NEXTVAL,  
 SYSTIMESTAMP,  
 CASE   
 WHEN INSERTING THEN 'INSERT'  
 WHEN UPDATING THEN 'UPDATE'  
 END,  
 'client\_id=' || :NEW.client\_id || ', status=' || :NEW.status  
 );  
EXCEPTION  
 WHEN OTHERS THEN  
 RAISE\_APPLICATION\_ERROR(-20001, 'Audit Log Failure: ' || SQLERRM);  
END;  
/

### 2. Package: State Management and Validation

The package body keeps track of the last\_inserted\_id in a private variable. The procedure handles the business logic for setting the status based on the score.

SQL

CREATE OR REPLACE PACKAGE client\_pkg AS  
 PROCEDURE add\_client(p\_full\_name IN VARCHAR2, p\_score IN NUMBER, p\_phone IN VARCHAR2 DEFAULT NULL);  
 FUNCTION get\_last\_client\_id RETURN NUMBER;  
END client\_pkg;  
/  
  
CREATE OR REPLACE PACKAGE BODY client\_pkg AS  
 gv\_last\_inserted\_id NUMBER;  
  
 PROCEDURE add\_client(p\_full\_name IN VARCHAR2, p\_score IN NUMBER, p\_phone IN VARCHAR2 DEFAULT NULL) IS  
 v\_status VARCHAR2(10);  
 BEGIN  
 -- Validation  
 IF p\_full\_name IS NULL THEN  
 RAISE\_APPLICATION\_ERROR(-20002, 'Full name cannot be NULL');  
 END IF;  
   
 IF p\_score < 0 OR p\_score > 100 THEN  
 RAISE\_APPLICATION\_ERROR(-20003, 'Score must be between 0 and 100');  
 END IF;  
  
 -- Status Logic  
 IF p\_score >= 60 THEN  
 v\_status := 'ACTIVE';  
 ELSE  
 v\_status := 'INACTIVE';  
 END IF;  
  
 gv\_last\_inserted\_id := seq\_client.NEXTVAL;  
  
 INSERT INTO clients (client\_id, full\_name, phone, score, status)  
 VALUES (gv\_last\_inserted\_id, p\_full\_name, p\_phone, p\_score, v\_status);  
 END add\_client;  
  
 FUNCTION get\_last\_client\_id RETURN NUMBER IS  
 BEGIN  
 RETURN gv\_last\_inserted\_id;  
 END get\_last\_client\_id;  
END client\_pkg;  
/

### 3. Dynamic SQL: Secure Row Count

This procedure uses DBMS\_ASSERT to prevent SQL injection before performing the dynamic count query.

SQL

CREATE OR REPLACE PROCEDURE get\_table\_row\_count (  
 p\_table\_name IN VARCHAR2,  
 p\_row\_count OUT NUMBER  
) IS  
 v\_safe\_name VARCHAR2(100);  
BEGIN  
 -- SQL Injection Protection  
 v\_safe\_name := DBMS\_ASSERT.SIMPLE\_SQL\_NAME(p\_table\_name);  
   
 EXECUTE IMMEDIATE 'SELECT COUNT(\*) FROM ' || v\_safe\_name INTO p\_row\_count;  
EXCEPTION  
 WHEN OTHERS THEN  
 RAISE\_APPLICATION\_ERROR(-20004, 'Error retrieving count for table ' || p\_table\_name || ': ' || SQLERRM);  
END;  
/

### 4. Function and Procedure: Business Logic

The function calculates the average, and the procedure applies it to the clients table. Note the custom exception handling for non-existent clients.

SQL

-- Function to get average score  
CREATE OR REPLACE FUNCTION get\_client\_avg\_score(p\_client\_id IN NUMBER)   
RETURN NUMBER IS  
 v\_avg NUMBER;  
BEGIN  
 SELECT AVG(score) INTO v\_avg   
 FROM credit\_checks   
 WHERE client\_id = p\_client\_id;  
   
 RETURN v\_avg; -- Returns NULL if no checks exist  
END;  
/  
  
-- Procedure to update status based on threshold  
CREATE OR REPLACE PROCEDURE update\_client\_status\_by\_avg(  
 p\_client\_id IN NUMBER,   
 p\_threshold IN NUMBER  
) IS  
 v\_avg\_score NUMBER;  
 v\_exists NUMBER;  
BEGIN  
 -- Check if client exists  
 SELECT COUNT(\*) INTO v\_exists FROM clients WHERE client\_id = p\_client\_id;  
 IF v\_exists = 0 THEN  
 RAISE\_APPLICATION\_ERROR(-20005, 'Client ID ' || p\_client\_id || ' does not exist.');  
 END IF;  
  
 v\_avg\_score := get\_client\_avg\_score(p\_client\_id);  
  
 IF v\_avg\_score IS NOT NULL THEN  
 UPDATE clients   
 SET status = CASE WHEN v\_avg\_score >= p\_threshold THEN 'ACTIVE' ELSE 'INACTIVE' END,  
 updated\_at = SYSTIMESTAMP  
 WHERE client\_id = p\_client\_id;  
 END IF;  
END;  
/

### 5. Collections and BULK Processing

This block demonstrates BULK COLLECT and iterating through a collection to classify clients based on their average scores.

SQL

DECLARE  
 TYPE t\_client\_ids IS TABLE OF clients.client\_id%TYPE;  
 v\_ids t\_client\_ids;  
 v\_avg NUMBER;  
 v\_class VARCHAR2(20);  
BEGIN  
 SELECT client\_id BULK COLLECT INTO v\_ids FROM clients;  
  
 IF v\_ids.COUNT = 0 THEN  
 DBMS\_OUTPUT.PUT\_LINE('No clients found.');  
 ELSE  
 FOR i IN 1..v\_ids.COUNT LOOP  
 v\_avg := get\_client\_avg\_score(v\_ids(i));  
   
 IF v\_avg IS NULL THEN  
 v\_class := 'NO\_DATA';  
 ELSIF v\_avg >= 80 THEN  
 v\_class := 'TOP';  
 ELSIF v\_avg >= 60 THEN  
 v\_class := 'OK';  
 ELSE  
 v\_class := 'RISK';  
 END IF;  
   
 DBMS\_OUTPUT.PUT\_LINE('Client ID: ' || v\_ids(i) || ' | Class: ' || v\_class || ' | Avg: ' || NVL(TO\_CHAR(v\_avg), 'N/A'));  
 END LOOP;  
 END IF;  
END;  
/

### 6. Explicit Cursor: Controlled Reporting

This block uses the manual OPEN, FETCH, and CLOSE lifecycle instead of a FOR loop to show complete control over the cursor.

SQL

DECLARE  
 CURSOR c\_clients IS   
 SELECT client\_id, full\_name, score FROM clients;  
   
 v\_id clients.client\_id%TYPE;  
 v\_name clients.full\_name%TYPE;  
 v\_score clients.score%TYPE;  
BEGIN  
 OPEN c\_clients;  
 LOOP  
 FETCH c\_clients INTO v\_id, v\_name, v\_score;  
 EXIT WHEN c\_clients%NOTFOUND;  
   
 DBMS\_OUTPUT.PUT\_LINE(  
 'ID: ' || v\_id ||   
 ' | Name: ' || v\_name ||   
 ' | Score: ' || NVL(TO\_CHAR(v\_score), 'NONE')  
 );  
 END LOOP;  
 CLOSE c\_clients;  
END;  
/

Would you like me to provide some test cases (INSERTs and procedure calls) to verify that all these components are working correctly together?